TEC | Tecnológico de Costa Rica

Tarea 3

Introducción y Taller de programación
I-Semestre 2022

Estudiante Marco Rodríguez
Estudiante Maximilian Latysh

Prof. Edgar Rojas

24 de marzo de 2022

1. Operaciones Básicas

1. Sucesor: Recibe un número entero n y retorna el siguiente entero después de este.

```
def sucesor(n):
    return n+1
```

2. Antecesor: Recibe un número entero n y retorna el antecesor de "n"numero.

```
def antecesor(n):
    ant = n - 1
    return ant
```

3. Cuadrado: Recibe un entero x retorna el cuadrado de este.

```
def cuadrado(x):
    cua = x**2
    return cua
```

4. Cero X2: Recibe 3 coeficientes correspondientes a una ecuación cuadrático retorna la segunda solución de la ecuación.

```
def cerox2(A,B,C,x2):
    x2 = -B-(B**2-4*A*C)**.5/(2*A)
    return(x2)
```

5. Mayor: Recibe dos números y retorna el mayor de estos dos.

```
def mayor(x,y):
    if x >= y:
        return x
    elif x < y:
        return y
```

6. Menor: Recibe dos números y retorna el menor de estos dos.

```
def menor(a, b):
    if a >= b:
        return b
    return a
```

7. Hipotenusa: Recibe la longitud de dos catetos retornando la hipotenusa de dos catetos.

```
def hipotenusa(c1,c2,hipot,resul):
    hipot = (c1**2 + c2**2)
    resul = hipot**0.5
    return resul
```

8. Desigualdad triangular: Recibe la longitud de tres catetos y retorna si es verdadera o falsa la desigualdad de un triangulo.

```
def desig(a,b,c):
    if a+b>c and b+c>a and a+c> b:
        return True
    else:
        return False
```

9. Desigualdad triangular 2: Comprueba si tres puntos en un plano están planteadas sobre la misma recta. Si sí, no son un triángulo.

```
# recibimos 3 tuplas como los puntos.
def des_tri_2(punto_1, punto_2, punto_3):
    pendiente_1 = None
    pendiente_2 = None
    if not punto_1[0] == punto_2[0]:
        pendiente_1 = (punto_2[1] - punto_1[1])/(punto_2[0] -
            punto_1[0])
    else:
        pendiente_1 = "x"
    if not punto_2[0] == punto_3[0]:
        pendiente_2 = (punto_3[1] - punto_2[1])/(punto_3[0] -
            punto_2[0])
    else:
        pendiente_2 = "x"
    return not pendiente_1 == pendiente_2
```

2. Ciclos

1. MCD: Recibe dos números enteros y encuentra el máximo común divisor de los dos. Este algoritmo consiste en probar cada número entre 2 y la raíz cuadrada del valor más pequeño y encontrar el más grande que cumple esta condición. Si no se encuentra ninguno que cumple esta condición, el programa retorna 1.

```
def menor(a, b):
    if a >= b:
        return b
    return a


def raiz_entera(n):
    if n < 1:
        return 0
    inf = 1
    sup = n
    mid = (inf + sup) >> 1
    cuadmid = mid*mid
    while inf + 1 != sup:
        if cuadmid > n:
            sup = mid
        elif cuadmid < n:
            inf = mid
        else:
            return mid
        mid = (inf + sup) >> 1
        cuadmid = mid*mid
    return mid


def a_natural(n):
    return n - 2*menor(n,0)


def MCD(a, b):
    a = a_natural(a)
    b = a_natural(b)
    if a == b:
        return a
    limite = raiz_entera(menor(a, b))
    respuesta = 1
    for i in range(2, limite):
        if not (a % i or b % i):
            respuesta = i
    return respuesta
```

2. mcm: Recibimos dos números enteros y encontramos el mínimo común divisor de los dos. Sabemos que el menor común múltiplo de los dos números va a tener la práctica unión entre los factores de cada uno. Por lo tanto, primero encontramos los factores de ambos números y después los comparamos multiplicándolos al final.

```
def factores(n):
    fac = 3
    factores = []
    if n == 0:
        return [0]
    if n < 0:
        factores.append(-1)
        n *= -1
    while not n % 2:
        n >>= 1
        factores.append(2)
    while 1:
        if n == 1:
            return factores
        elif fac*fac > n:
            return factores + [n]
        elif n % fac:
            fac += 2
        else:
            factores.append(fac)
            n //= fac


def mcm(a, b):
    factores_a, factores_b = factores(a), factores(b)
    factores_aux = []
    for i in factores_a:
        factores_aux.append(i)
        if i in factores_b:
            factores_b.remove(i)
    factores_aux = factores_aux + factores_b
    mcm = 1
    print(factores_aux)
    for i in factores_aux:
        mcm *= i
    return mcm
```

3. Suma de cubos: Recibimos un número natural y encontramos la suma de sus cubos.

```
def Suma_de_Cubos(n):
    suma=0
    for n in range(1,n+1):
        suma+=n**3
    return suma
```

4. Es primo? Recibe un número natural y comprueba si es primo o no utilizando un simple método iterativo aprendido por medio de las asignaciones. En esencia, contamos los factores del número dado y (a diferencia de que sea 1 o 0), si la cantidad de factores primos que encontramos es 1, sabemos que tenemos un primo.

```
def es_primo(n):
    fac = 3
    if n == 0 or n == 1:
        return False
    total = 0
    while not n % 2:
        n >>= 1
        total += 1
    while 1:
        if n == 1:
            break
        elif fac*fac > n:
            total += 1
            break
        elif n % fac:
            fac += 2
        else:
            total += 1
            n //= fac
    if total == 1:
        return True
    return False
```

5. n-simo primo: Recibe el índice + 1 del primo deseado más el limite bajo el cual se puede encontrar este primo. Retornamos el valor del primo en este índice. Esta calculación la realizamos utilizando la criba de Erastótenes.

```
def nsimo_primo(n, limite):
    criba = [not (i % 2) for i in range(limite - 1)]
    criba[0] = 0
    primo = 3
    cuad = 9
    if n == 1:
        return 2
    contador = 2
    while primo < limite and contador < n:
        contador += 1
        cuad -= 2
        for i in range(1 + (limite - cuad - 2) // primo):
            criba[cuad + primo*i] = 1
        primo += 1
        while criba[primo - 2]:
            primo += 1
        cuad = primo*primo
    return primo
```

6. Cantidad de dígitos: Recibe un número real y retorna la cantidad de dígitos enteros que posee.

```
def cantidad_digitos(n):
    cantidad = 0
    while n > 1 or n < -1:
        cantidad += 1
        n //= 10
    return cantidad
```

7. Invertir un número: Recibe un número entero y encuentra la reversa de sus dígitos.

```
def invertir(n):
    invertido = 0
    while n != 0:
        invertido = invertido*10+n%10
        n //= 10
    return invertido
```

8. Factorial: Recibe un número natural y retorna su factorial.

```
def factorial(n):
    factorial = 1
    for i in range(1, n + 1):
        factorial = factorial * i
    return factorial
```

9. Suma de fracciones: Recibe un número natural y retorna el valor con el criterio dado.

```
def suma_fracciones(n):
    suma=0
    for i in range(1,n+1):
        suma+=((-1)**i)/i
    return suma
```

10. Suma de suma: Recibimos un número n y retornamos la suma de la suma de los productos de i y j mientras cambian.

```
def suma_suma(n):
    suma = 0
    for i in range(1, n + 1):
        temp_suma = 0
        for j in range(1, i + 1):
            temp_suma += j
        suma += temp_suma*i
    return suma
```

11. Producto: Recibimos un número natural y encontramos el valor bajo el criterio solicitado. Aunque parece que se necesita utilizar un for-loop, podemos calcular el valor utilizando una suma de Gauss.

```
def aplicar_potencia(n, pot):
    respuesta = 1
    for i in range(pot):
        respuesta *= n
    return respuesta


def producto(n):
    return 1/aplicar_potencia(2, n*(n + 1)//2)
```

12. Producto de sumas: Recibe un número natural n y retorna el producto de su suma bajo el criterio $2 * i/j$

```
def producto_sumas(n):
    mult = 0
    for i in range(2, n + 1):
        temp_suma = 0
        for j in range(1, i*i + 1):
            temp_suma += 1/j
        mult *= temp_suma*i*2
    return suma
```

13. Números perfectos: Recibimos un número natural y comprobamos sí es perfecto o no. Para esto, calculamos todos sus factores y realizamos una suma de estos mismos.

```
def factores(n):
    fac = 2
    original = n
    factores = [1]
    if n == 0 or n == 1:
        return factores
    while 1:
        if n == 1:
            break
        elif fac > n:
            break
        elif n % fac:
            fac += 1
        elif fac != original:
            factores.append(fac)
            fac += 1
        else:
            break
    if original in factores:
        factores.remove(original)
    return factores


def numeros_perfectos(n):
    factores_n = factores(n)
    suma = 0
    for i in factores_n:
        suma += i
    return suma == n
```

14. Números amigos

a) Comprueba la amigabilidad de dos números a y b sacando los factores de ambos, realizando dos sumas separadas y finalmente comparándolos uno con el otro.

```
def factores(n):
    fac = 2
    original = n
    factores = [1]
    if n == 0 or n == 1:
        return factores
    while 1:
        if n == 1:
            break
        elif fac > n:
            break
        elif n % fac:
            fac += 1
        elif fac != original:
            factores.append(fac)
            fac += 1
        else:
            break
    if original in factores:
        factores.remove(original)
    return factores


def amigos(a, b):
    factores_a = factores(a)
    factores_b = factores(b)
    suma_a = 0
    suma_b = 0
    for i in factores_a:
        suma_a += i
    for i in factores_b:
        suma_b += i
    return suma_a == b and suma_b == a
```

b) Encuentra el amigo de n sacando sus factores, realizando una suma de estos factores, sacando los factores de esta suma y comprobando si esta última suma es igual a n.

```
def factores(n):
    fac = 2
    original = n
    factores = [1]
    if n == 0 or n == 1:
        return factores
    while 1:
        if n == 1:
            break
        elif fac > n:
            break
        elif n % fac:
            fac += 1
        elif fac != original:
            factores.append(fac)
            fac += 1
        else:
            break
    if original in factores:
        factores.remove(original)
    return factores


def amigo(n):
    factores_n = factores(n)
    suma = 0
    for i in factores_n:
        suma += i
    factores_suma = factores(suma)
    suma_suma = 0
    for i in factores_suma:
        suma_suma += i
    return suma if suma_suma == n else -1
```

15. Conjetura de Goldbach: Encontramos los dos primos que sumados nos proporcionan el par n. Realizamos esta calculación intentando verificar cada par de números que suman a n a si ambos valores de la pareja son primos.

```
def es_primo(n):
    fac = 3
    if n == 0 or n == 1:
        return False
    total = 0
    while not n % 2:
        n >>= 1
        total += 1
    while 1:
        if n == 1:
            break
        elif fac*fac > n:
            total += 1
            break
        elif n % fac:
            fac += 2
        else:
            total += 1
            n //= fac
    if total == 1:
        return True
    return False


def goldbach_r(original, supuesto):
    candidato1 = supuesto-1
    candidato2 = original - candidato1
    while not es_primo(candidato1) or not es_primo(candidato2):
        supuesto -= 2
        candidato1 = supuesto
        candidato2 = original - candidato1
    return [candidato1, candidato2]


def goldbach(n):
    if n == 0:
        return [0,0]
    if n == 2:
        return [1,1]
    return goldbach_r(n, n - 1)
```

16. Cuatro cuadrados: Recibe un número n e intenta encontrar todos los 4 cuadrados que sumados proporcionan n. Para calcular estos cuatro cuadrados, el programa intenta de forma iterativa ver si un cuarteto de valores proporcionan n. Sino, nada más se intenta con un otro cuarteto de valores hasta intentarlos todos. Además, existen modificaciones para evitar repeticiones y gasto inútil de tiempo.

```
def raiz_entera(n):
    if n < 1:
        return 0
    inf = 1
    sup = n
    mid = (inf + sup) >> 1
    cuadmid = mid*mid
    while inf + 1 != sup:
        if cuadmid > n:
            sup = mid
        elif cuadmid < n:
            inf = mid
        else:
            return mid
        mid = (inf + sup) >> 1
        cuadmid = mid*mid
    return mid


def cuatroCuadrados(n):
    respuesta = []
    raiz = raiz_entera(n) + 1
    for i in range(0, raiz):
        for j in range(i, raiz):
            for o in range(j, raiz):
                for p in range(o, raiz):
                    if (i*i + j*j + o*o + p*p == n):
                        posible = [i, j, o, p]
                        respuesta.append(posible)
    return respuesta
```

17. Teorema de Carmichael: Recibe un número natural n mayor que 12 y retorna el menor primo que ningún otro número de fibonacci anterior comparte. Para este propósito, se toma en cuenta en algoritmo para generar los valores de fibonacci (asumimos que el primer fibonacci es 0, el segundo es 1 y así en adelante. De esta forma, el fibonacci en posición 13 es 144) y un método para calcular los factores de un dado valor. Después, eliminamos los factores que comparten los valores previos de fibonacci. Si en algún caso se eliminan todos los factores, retornamos -1. Sino, retornamos el factor primo más pequeño que no comparte con ningún otro valor de fibonacci previo.

```
def fibonaccis(n):
    fibonacci = [0, 1]
    for i in range(0, n - 2):
        fibonacci.append(fibonacci[-1] + fibonacci[-2])
    return fibonacci


def raiz_natural(n):
    inf = 1
    sup = n
    mid = (inf + sup) >> 1
    cuadmid = mid*mid
    while inf + 1 != sup:
        if cuadmid > n: sup = mid
        elif cuadmid < n: inf = mid
        else: return mid
        mid = (inf + sup) >> 1
        cuadmid = mid*mid
    return mid


def generar_criba(limite):
    limite_raiz = raiz_natural(limite)
    criba = [not (i % 2) for i in range(limite_raiz - 1)]
    criba[0] = 0
    factores = [2]
    primo = 3
    cuad = 9
    while primo < limite_raiz:
        factores.append(primo)
        if cuad <= limite_raiz:
            cuad -= 2
            for i in range(1 + (limite_raiz - cuad - 2) // primo):
                criba[cuad + primo*i] = 1
        primo += 1
        while primo - 2 < len(criba):
            if criba[primo - 2]:
                primo += 1
```

```
            else:
                break
        cuad = primo*primo
    return factores


def factorizar(n, criba):
    factores = []
    pos = 0
    if n == 0 or n == 1:
        return [1]
    while pos < len(criba):
        if n % criba[pos]:
            pos += 1
        else:
            if criba[pos] not in factores:
                factores.append(criba[pos])
            n //= criba[pos]
    if len(factores) == 0 or n != 1:
        return factores + [n]
    return factores


def carmichael(n):
    fibonacci_n = fibonaccis(n)
    criba = generar_criba(fibonacci_n[-1])
    factores_n = factorizar(fibonacci_n[-1], criba)
    if len(factores_n) == 1:
        return factores_n[0]
    factores_previos = (1, 2, 3, 5, 7, 11, 13, 17, 89)
    for i in factores_previos:
        while i in factores_n:
            factores_n.remove(i)
    for i in range(13,n):
        for j in factorizar(fibonacci_n[i - 1], criba):
            if j in factores_n:
                factores_n.remove(j)
    if len(factores_n) == 0:
        return -1
    return factores_n[0]
```

Referencias

[1] E. R. Jiménez, *Fundamentos de Programación.* 2022.